European Scientific Journal May 2017 edition Vol.13, No.15 ISSN: 1857 – 7881 (Print) e - ISSN 1857- 7431

Enhancement of Web Security Against External Attack

Md. Fazlul Haque
Mohammad Badrul Alam Miah
Fuyad Al Masud
Department of Information and Communication Technology,
Mawlana Bhashani Science and Technology University,
Santosh-1902, Tangail, Bangladesh.

doi: 10.19044/esj.2017.v13n15p228 URL:http://dx.doi.org/10.19044/esj.2017.v13n15p228

Abstract

The security of web-based services is currently playing a vital role for the software industry. In recent years, many technologies and standards have emerged in order to handle the security issues related to web services. This paper shows techniques to enhance the security of web services, and some of the recent challenges and recommendations of a proposed model to secure web services. It shows the security process of a real life web application, which includes; HTML5 forms, login security, and a single sign-on solution. This paper also aim to discuss the ten (10) most common web security vulnerabilities and how to prevent the web application from three (3) of the vulnerabilities. Amongst them are; SQL Injection, Cross Site Scripting and Broken Authentication, and Session Management.

Keywords: Security, External Attack, Vulnerability

Introduction

The world wide web is largely based on Hypertext Transfer Protocol (HTTP), which specifies the format of message exchanged by a client known, in this case, as the user agent and the server. However, the request format which is most familiar to internet users is what is commonly called the Uniform Resource Locator (URL). When a user enters a URL into the browser window, the browser first checks the scheme of the URL to determine the protocol. Terms like web application, website, web-based system, web-based software, or simply web may have the same meaning. The web is indispensable in modern commerce, entertainment, and social interaction. It is a complex delivery platform for sophisticated distributed applications with multifaceted security requirements.

However, most web browsers, servers, network protocols, browser extensions, and their security mechanisms were designed without analytical foundations. In complicating matters further, the web continues to evolve with new browser features, protocols, and standards which were added at a rapid pace. The specifications of new features are often complex, lack clear threat models, and involves unstated and unverified assumptions about other components of the web. As a result, new features can introduce new vulnerabilities and break security invariants assumed by web applications.

In this paper, we take the first step towards building a comprehensive formal foundation for web security. Also, we discuss the common web security vulnerabilities and how to prevent the web application from the first three vulnerabilities, and then implementing those vulnerabilities in real life web application.

Consequently, there is a large number of work on formally verifying security properties of network protocols, web security including model checking using a variety of tools (Mitchell et al., 1997), and constraint-based methods (Millen & Shmatikov, 2001). Based on the study of SQL Injection Attacks and Countermeasures (Sayyed Mohammad et al., 2013), this paper gives scientific categorization of strategies to avert and recognize SQLIA. We characterize web application vulnerabilities and how they may bring about SQLIA. At a point, we show an order of SQLIA in view of its weakness. Sonoda et al. (2016), in their paper, show the Maximum Likelihood Estimation in Stochastic Model of SQL Injection Attacks. In the paper titled "SQL Injection Detection and Prevention using Pattern Matching Algorithm", they try to use Pattern Matching Algorithm to detect and prevent SQLI attacks on websites, hence providing maximum security to websites (Kharche et al., 2015).

Gupta et al. (2016) in their paper show a context-sensitive sanitization based on XSS defensive framework for the cloud environment. It discovers all the hidden injection points in HTML5-based web applications deployed on the platforms of cloud. Furthermore, it sanitizes the XSS attack payloads injected in such points in a context-sensitive manner.

The Organization of this Paper: The remainder of this paper is organized as follows:

- The Common Models section represents the existing models and related reference papers.
- The Most Common Vulnerabilities section shows the ten common vulnerabilities.
- The Implementation section implements our proposed model.

Common Models

There are many threats associated with web browsing, web applications and web security. They include phishing, drive-by downloads, blog spam, account takeover, and click fraud. Although some of these threats revolve around exploiting implementation vulnerabilities such as memory safety errors in browsers or tricking the user. Our focus in this paper is on the ways in which an attacker can abuse web functionality that exists by design. For example, an HTML form element allows a malicious website generate GET and POST requests to arbitrary web sites, leading to security risks like Cross-Site Request Forgery (CSRF). Websites use a number of different strategies to defend themselves against CSRF (Barth et al., 2008). However, we lack a scientifically rigorous methodology for studying these defenses. By formulating an accurate model of the web, we can evaluate the security of these defenses and determine how they interact with extensions to the web platform.

The core idea in our model is to describe what could occur if a user navigates the web and visits sites in ways the web was designed to be used. For example, the user could choose to type any web address into the address bar and visit any site, or click on a link provided by one site to visit another. Since browsers support the "back" button, returning the user to a previously visited page, many sites in effect allow a user to click on all of the links presented on a page, and not just one. When the user visits a site, the site could serve a page with any number of characteristics, possibly setting a cookie, or redirecting the user to another site. The set of events that could occur, therefore, includes; browser requests, responses, cookies, redirects, and so on, transmitted over HTTP or HTTPS. Therefore, we believe that examining a set of possible events accurately captures the way web security mechanisms are designed. For example, the web is designed to allow a user to visit a good site in one window and a potentially malicious site in another. Since the back button is so popular, web security mechanisms are usually designed to be secure even if the user returns to a previously visited page and progresses differently the second (or third or fourth) time.

The Most Common Vulnerabilities

There is a paper in which there are Tracing known as security vulnerabilities in software repositories, and it uses a Semantic Web-enabled modeling approach (Alqahtani et al., 2016). Thus, the common vulnerabilities which we got in real applications are:

i) SQL Injection: Injection is a security vulnerability that allows an attacker to alter back-end SQL statements by manipulating the user supplied data. Injection occurs when the user input is sent to an interpreter as part of a

command or query to trick the interpreter into executing unintended commands and gives access to unauthorized data.

The SQL command which when executed by web application can also expose the back-end database.

ii) Cross Site Scripting: Cross Site Scripting is shortly known as XSS. XSS vulnerabilities target scripts embedded in a page that are executed on the client side i.e. user browser rather then at the server side. These flaws can occur when the application takes untrusted data and send it to the web browser without proper validation. Attackers can use XSS to execute malicious scripts on the users in the case of victim browsers. Since the browser cannot know if the script is trustworthy or not, the script will be executed, and the attacker can hijack session cookies, deface websites, or redirect the user to unwanted and malicious websites. XSS is an attack which allows the attacker to execute the scripts on the victim's browser.

iii) Broken Authentication and Session Management: The websites usually create a session cookie and session ID for each valid session, and these cookies contain sensitive data like username, password, etc. When the session is ended either by logout or the browser closed abruptly, these cookies should be invalidated i.e. for each session, there should be a new cookie.

If the cookies are not invalidated, the sensitive data will exist in the system. For example, for a user using a public computer (Cyber Cafe), the cookies of the vulnerable site sits on the system and is exposed to an attacker. When an attacker uses the same public computer, the sensitive data would be compromised after some time.

In the same manner, a user using a public computer, instead of logging off, he closes the browser abruptly. When an attacker uses the same system, when browsing the same vulnerable site, the previous session of the victim will open up. Then, the attacker can do whatever he wants to do like stealing profile information, credit card information, etc.

iv) Insecure Direct Object References: It occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key as in URL or as a FORM parameter. The attacker can use this information to access other objects and can create a future attack to access the unauthorized data.

v) Cross Site Request Forgery: Cross Site Request Forgery is a forged request from the cross site. CSRF attack is an attack that occurs when a malicious website, email, or program causes a user's browser to perform an unwanted action on a trusted site for which the user is currently authenticated. A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web

application. A link will be sent by the attacker to the victim. When the user clicks on the URL when logged into the original website, the data will be stolen from the website.

vi) Security Misconfiguration: Security Configuration must be defined and deployed for the application, frameworks, application server, web server, database server, and platform. If these are not properly configured, an attacker can have unauthorized access to sensitive data or functionality. Sometimes, such flaws result in complete system compromise. Keeping the software up to date is also a form of good security.

vii) Insecure Cryptographic Storage: Insecure Cryptographic storage is a common vulnerability which exists when sensitive data is not stored securely. The user's credentials, profile information, health details, credit card information, etc. comes under sensitive data information on a website. These data will be stored on the application database. When these data are stored improperly by not using encryption or hashing*, it will be vulnerable to the attackers. (*Hashing is transformation of the string characters into shorter strings of fixed length or a key. To decrypt the string, the algorithm used to form the key should be available)

viii) Failure to Restrict URL Access: Web applications check URL access rights before rendering protected links and buttons. Thus, applications need to perform similar access control checks each time these pages are accessed. In most of the applications, the privileged pages, locations, and resources are not presented to the privileged users. By an intelligent guess, an attacker can access privilege pages, access sensitive pages, invoke functions, and view confidential information (Shinde et al., 2016).

ix) Insufficient Transport Layer Protection: This deals with information exchange between the user (client) and the server (application). Applications frequently transmit sensitive information like authentication details, credit card information, and session tokens over a network. By using weak algorithms or using expired or invalid certificates or not, using SSL can allow the communication to be exposed to untrusted users. This may result to the compromise of a web application and/or the stealing of sensitive information (Duncan et al., 2016).

x) Unvalidated Redirects and Forwards: The web application uses few methods to redirect and forward users to other pages for an intended purpose. If there is no proper validation while redirecting to other pages, attackers can make use of this and can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Implementation of the Proposed Model

In this section, we described our proposed model and showed the first three common vulnerabilities effect and how to protect them. Here, we will cover the SQL Injection, Cross Site Scripting, and Broken Authentication and Session Management.

i) SQL Injection Attacks

SQL injection attacks are caused by an unexpected user input for the developer of the web application (Shinde et al., 2016). The method of attack is very simple, but the power of SQL injection attacks is very strong. However, it causes a serious social problem. Now, we will give an example of SQL injection attack. Assume that there exists a web application that has the following login authentication program as stated below:

SELECT * FROM users WHERE id=’userID’ AND pw=’pass’

From the above example, if malicious user input ’ OR 1=1-- into the form of id, then the following SQL sentence is constructed by the web application.

SELECT * FROM users WHERE id=’’ OR 1=1--’ AND pw=’pass’

1 = 1 is always true and the string -- deauthorizes the following strings.

’ AND pw=’pass’

Therefore, if this attack were successful, then malicious user can gain unauthorized access to the database of the web application.

In our proposed model, we tried the following SQL injection queries.

Executed SQL query when username is userID and password is a single quote:

SELECT * FROM users WHERE name=userID and password='"

Executed SQL query when username is userID and password is ' or '1'='1:

SELECT * FROM users WHERE name='userID' and password='' or '1'='1'

Table 1. The Cheat Sheet for checking the SQL Injection in our model

username	password	SQL Query
userID	userID	SELECT * FROM users WHERE name='userID' and password='userID'
userID	' or '1'='1	SELECT * FROM users WHERE name='userID' and password='' or '1'='1'
userID	' or 1='1	SELECT * FROM users WHERE name='userID' and password='' or 1='1'
userID	1' or 1=1 -- -	SELECT * FROM users WHERE name='userID' and password='' or 1=1-- -'
' or '1'='1	' or '1'='1	SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'

' or ' 1=1	' or ' 1=1	SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'
1' or 1=1 -- -	blah	SELECT * FROM users WHERE name='1' or 1=1 - -' and password='blah'

Furthermore, we used the SQL Queries from the Table 1 Cheat Sheet on our proposed model architecture, and finally prevented the SQL injection. We have seen few SQL injection-preventing algorithm. Also, we found that our proposed architecture is the simple and easiest to implement in the web application. In Figure 1, we show the proposed model architecture for SQL injection. Here, we used three input text fields for login to the database. We used an auto-generated Captcha, which prevents our web application from the robotic request. In SQL query generation section, we placed all the queries from our cheat sheet and then checked the coding level condition. Finally, we are trying to match the condition on Matching query section. After putting all the SQL queries, we put whitelisting in our HTML text Field. Even though the query is successful, we still put whitelisting in our HTML text Field. The most important thing in our evaluation is that we do not show any error or success message anywhere in the Web Application.

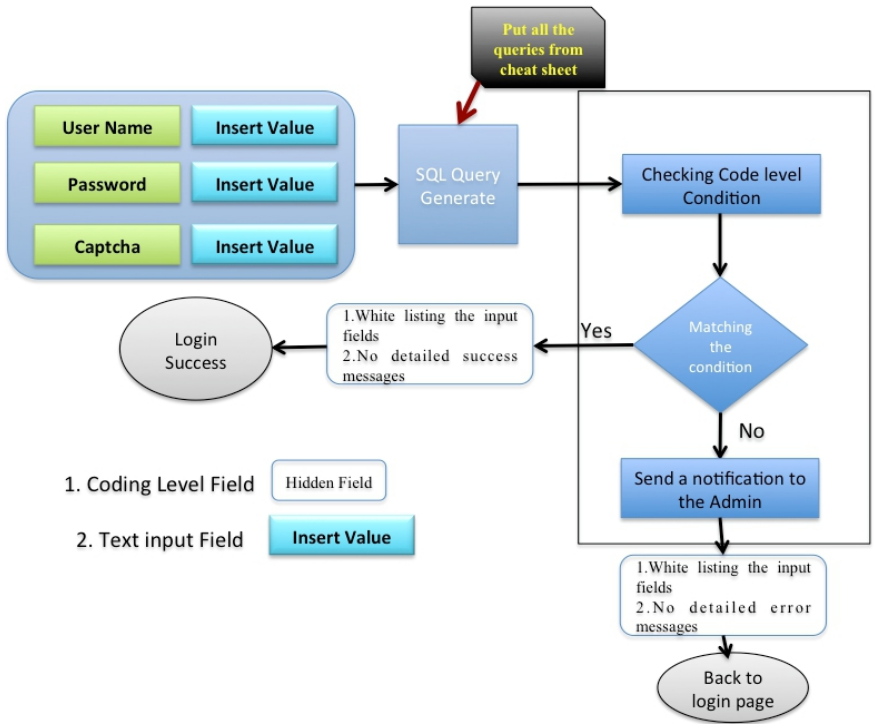

Figure 1. The proposed model architecture for SQL injection

ii) Cross Site Scripting: Cross Site Scripting is shortly known as XSS. XSS vulnerabilities target scripts embedded in a page that is executed on the client side i.e. user browser rather than at the server side. These flaws can occur when the application takes untrusted data and send it to the web browser without proper validation. Attackers can use XSS to execute malicious scripts on the users in the case of victim browsers. Since the browser cannot know if the script is trustworthy or not, the script will be executed. Also, the attacker can hijack session cookies, deface web sites, or redirect the user to unwanted and malicious websites. XSS is an attack which allows the attacker to execute the scripts on the victim's browser.

In 2016, Shashank Gupta and B.B.Gupta conducted a survey on the various journals on "Cross Site Scripting attacks and Defense mechanism" (Shashank Gupta et al., 2016). They analyzed the major concerns for web applications and Internet-based services which are persistent in several web applications. They highlighted some of the serious vulnerabilities found in the modern web applications. Below is our Proposed Architecture for Cross Site Scripting:

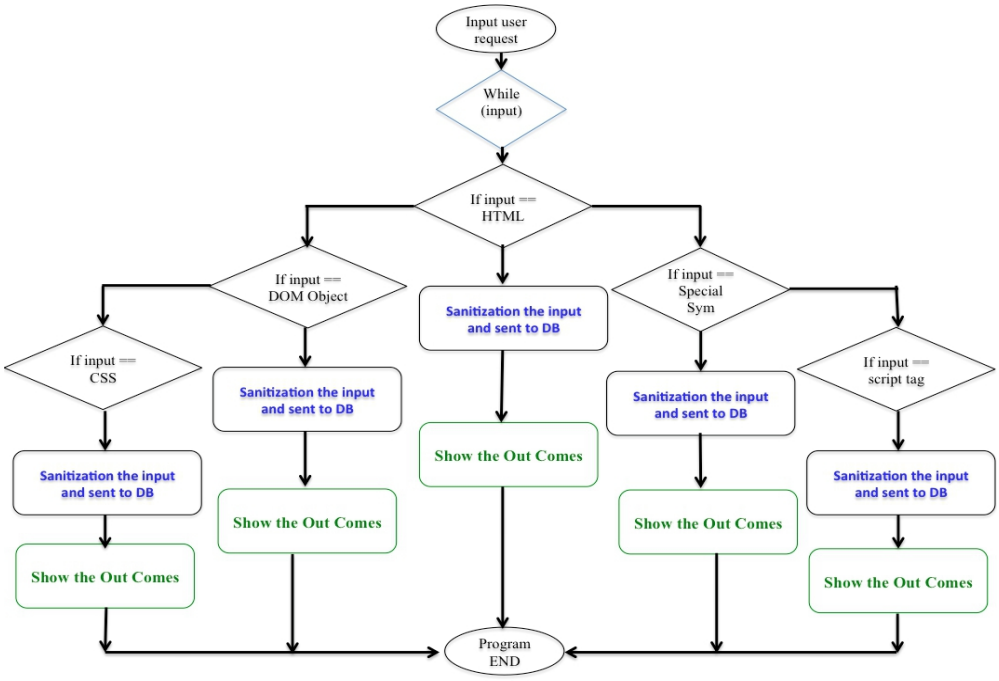

Figure 2. The proposed model architecture for Cross Site Scripting

For an attack to happen, the attacker tries to find the user input areas. The user input is given such priority because it is the only way for the user or client to interact with the server. If the attacker can be successful in injecting the malicious code into the server, an attack is guaranteed to happen. In order

to prevent the attacker to have that privilege, we sanitize the user input. As shown in Figure 2, we initially considered the user input. If the user input contains any HTML-specific tags like "<i>, <br>, <a> etc..", we sanitize the request and store it in the database. If the user input contains any special symbols which are generally used in script functions, they should be sanitized. If the user input contains any script tags which are one of the most important ways an attack is made possible, they should be properly sanitized. If the user input contains any styling related code, then the code should be filtered and stored in the database. Finally, we have restricted the redirection of a specific web application page to some other page through which we can stop most of the attacks. This can be done by sanitizing the user input if it contains any window location or document refferer methods. If the above methods are not followed, the attacker tries to steal the valuable information of the users like cookies. Usually, if we consider any login page, example sessions will be created for every user. The flaw in any browser is that it stores the session id in the form of a cookie. So, if the attacker steals this cookie, he can enter into the web application as an authorized user and the results can be more devastating.

iii) Broken Authentication and Session Management: The websites usually create a session cookie and session ID for each valid session, and these cookies contain sensitive data like username, password, etc. When the session is ended either by logout or the browser closed abruptly, these cookies should be invalidated i.e. for each session, there should be a new cookie.

If the cookies are not invalidated, the sensitive data will exist in the system.

A check should be done to find the strength of the authentication and session management. Keys, session tokens, and cookies should be implemented properly without compromising passwords.

Vulnerable Objects

- Session IDs exposed on URL can lead to session fixation attack.
- Session IDs same, before, and after logout and login.
- Session Timeouts are not implemented correctly.
- Application is assigning same session ID for each new session.
- Authenticated parts of the application are protected using SSL, and passwords are stored in hashed or encrypted format.
- The session can be reused by a low privileged user.

Implications

• By making use of this vulnerability, an attacker can hijack a session, and gain unauthorized access to the system which allows disclosure and modification of unauthorized information.

• The sessions can be highjacked using stolen cookies or sessions using XSS.

Proposed Model Architecture for Broken Authentication and Session Management

In this section, we focus on previously known session vulnerability. Also, we proposed the architecture which is shown in Figure 3, and our implementing login page shown in Figure 4. When any user or attacker tries to login into the admin page, they will need to put a username, password, and an auto-generated captcha.

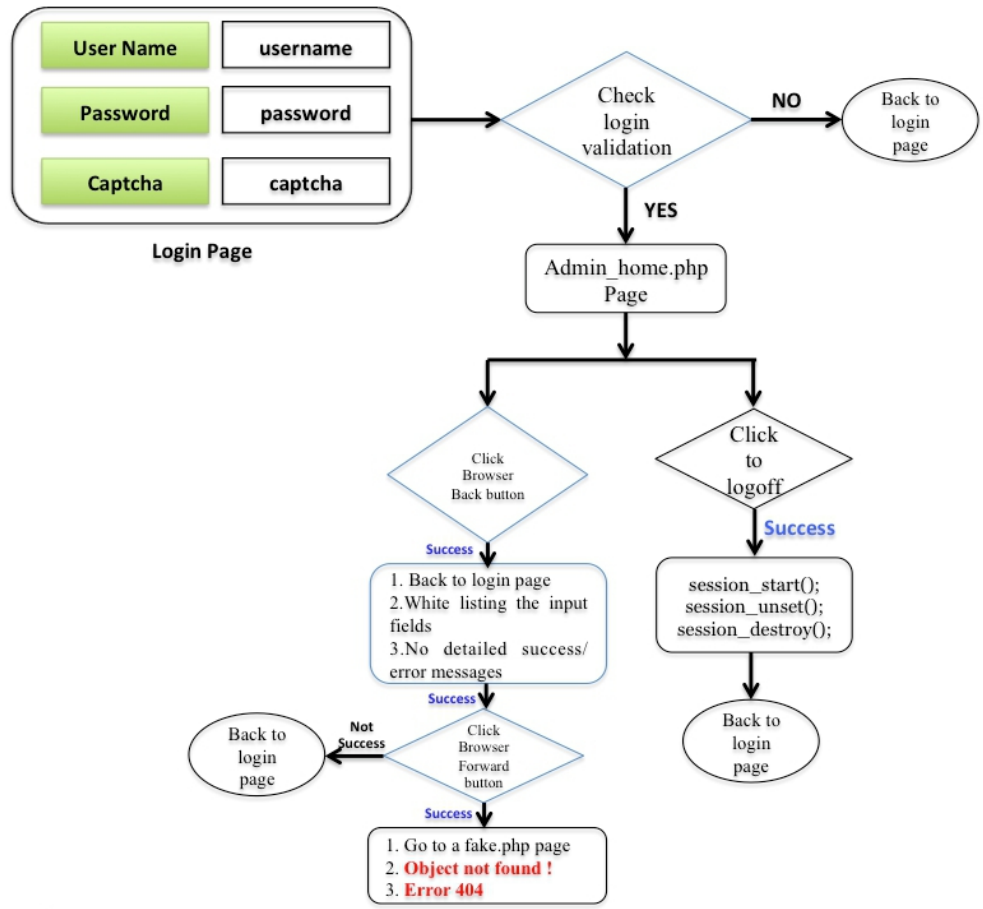

Figure 3. The proposed model architecture for Session Management

Figure 4. The login page of our implementing website

If all the fields matches with the database, then they can login or show the admin page. Now there are two options to hijacking the session; One is the logout or logoff section and another is the browser back and forward button section. Here, we made the most important security. We also made a fake.php page and linked it to our admin home page. If anyone can try the browser forward and the back button, then he can see the fake.php page. Actually, we placed there a condition which checks the value of the username and password. If anyone of these is empty, then it automatically shows the fake page with Object not found! and Error 404.

On the other section logoff option, we made an awesome security. If we tried to log out, we will have to execute the following code:

```
<?php
error_reporting(0);
session_start();
session_unset();
session_destroy();
?>
```

The **session_unset()** function frees all session variables currently registered, and the **session_destroy()** destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie. To use the session variables again, session_start() has to be called. Through this way, we can prevent our web application from Session hijacking.

Conclusion

In this paper, we tried to focus on the most common vulnerabilities of recent time. Also, we proposed individual prevention architecture for 'SQL Injection', 'Cross Site Scripting', and 'Broken Authentication and Session Management'. On the other hand, by implementing our proposed models, we have successfully prevented a real life website from those external attacks. Finally, it can be concluded that we have partially enhanced the web security

against recent external attacks. Of course, our implemented model does not capture the entire web platform and web security concerns.

References:

1. Alqahtani, Sultan S., Ellis E. Eghan, and Juergen Rilling (2016). "Tracing known security vulnerabilities in software repositories–A Semantic Web enabled modeling approach." Science of Computer Programming 121: 153-175.
2. Barth, Adam, Collin Jackson, and John C. Mitchell (2008). "Robust defenses for cross-site request forgery." Proceedings of the 15th ACM conference on Computer and communications security. ACM.
3. Duncan, Bob, Andreas Happe, and Alfred Bratterud (2016). "Enterprise IoT security and scalability: how unikernels can improve the status Quo." Proceedings of the 9th International Conference on Utility and Cloud Computing.
4. Gupta, Shashank, and B. B. Gupta (2016). "CSSXC: Context-sensitive Sanitization Framework for Web Applications against XSS Vulnerabilities in Cloud Environments." *Procedia Computer Science* 85: 198-205.
5. Kharche, Swapnil, Kanchan Gohad, and Bharti Ambetkar (2015). "Preventing SQL Injection attack using pattern matching algorithm." arXiv preprint arXiv:1504.06920.
6. Millen J. and Shmatikov V. (2001). "Constraint solving for boundedprocess cryptographic protocol analysis," in CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security. New York, NY, USA: ACM, pp. 166–175.
7. Mitchell J., Mitchell M., and U. Stern (1997). "Automated analysis of cryptographic protocols using Mur'," in Proc. IEEE Symp. Security and Privacy, pp. 141–151.
8. Sayyed Mohammad Sadegh Sajjadi and Bahare Tajalli Pour (2013). "Study of SQL Injection Attacks and Countermeasures", International Journal of Computer and Communication Engineering, Vol. 2, No. 5, September 2013
9. Shinde, Prashant S., Shrikant B. Ardhapurkar, and P. G. Scholar (2016). "Design and Implementation of VAPT Tool for Cyber Security Analysis using Response Analysis." International Journal of Engineering Science 4150.
10. Sonoda, Michio, Takeshi Matsuda, and Daiki Koizumi (2016). "On the approximate maximum likelihood estimation in stochastic model of SQL injection attacks." Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference.